Part of Speech Tagging for English Text Data

Jana Diesner
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
diesner@cs.cmu.edu

Abstract

A variety of Natural Language Processing (NLP) tasks, such as named entity recognition, stemming and question answering, benefit from knowledge of the words syntactic categories or Part-of-Speech (POS) [4][6]. POS taggers have been successfully applied to assign a single best POS to every word in a corpus [2][5][12]. This paper reports on the implementation and empiric comparison of three supervised, stochastic tagging approaches (Unigram Model, Hidden Markov Model, Viterbi algorithm). The presented comparison not only quantifies the tagging accuracy achieved by the Viterbi algorithm (93.9% on average), but also determines the partial accuracy gain that different components represented in Viterbi account for.

1 Introduction

While many words can be unambiguously associated with one POS or tag, e.g. noun, verb or adjective, other words match multiple tags, depending on the context that they appear in. *Wind* for example is a noun in the context of weather, and is a verb that refers to coiling something. [3] reports that in the Brown corpus, which is used in this study and describe in section 3.1 in detail, over 40% of the words are syntactically ambiguous. Thus, ambiguity resolution is the key challenge in tagging.

Taggers can be divided into systems that are rule-based, stochastic and transformation-based [6]. Stochastic taggers exploit the power of probabilities and machine learning techniques in order to disambiguate and tag sequences of words [10]. One widely and successfully applied approach to statistical modeling is Hidden Markov Models (HMM) [1]. In the domain of speech recognition, HMM has become the favored, state of the art model [8]. HMM are also used for tagging, where the most accurate systems achieve errors rates of less than four percent [4]. Most of the existent HMM taggers are trained with labeled data (e.g. [3][12]), while fewer ones use unlabeled data to train a HMM tagger based on expectation maximization (EM) (e.g. [5]). Based on the applicability of stochastic approaches for syntactical tagging and the potentially low error rates, I decided to pursue this

venue for this project. This work is furthermore motivated by my current research on semantic network analysis, where I expect knowledge about each word's POS to facilitate ontological coding.

2 Method

Markov Models (MM) model the probabilities of a linear sequence of non-independent events. Applying MM to tagging aims to find the most likely sequence of POS on the sentence level. Not assuming independence among the observations enables us to account for the fact that the words in a sentence may depend on each other, especially in the case of meaningful N-grams, such as *Department of Labor*. MM are applicable if the limited horizon assumption (future elements in a sequence are conditionally independent of past elements, given the present element) and the time invariance assumption (probabilities are stationery) hold truth [4][6]. Note that the latter assumption is a theoretical one; in practice, language is a dynamic system, in that rules (syntax) and elements (vocabulary) emerge and vanish over time and across places. Relating the outlined assumptions to tagging empowers us to exploit and combine every word's probability and context, as given by a word's predecessor(s), if available in training data. HMM, a probabilistic function of MM, brings these two pieces of information together by finding the tag sequence that maximizes the likelihood of the product of word probability (P(word | tag)) and tag sequence probability (P(tag | previous n tags)).

In tagging, the true sequence of POS that underlies an observed piece of text is unknown, thus forming the hidden states. The learner aims to find the sequence of hidden states that most probably has generated the observed sequence. This task is referred to as decoding, which means that given a set of observations (X) and a model (μ), we want to reveal the underlying Markov chain that is probabilistically linked to the observed states. Model μ consists of three parameters [8]:

1. Initial state probabilities (π). This is a vector that quantifies the probability of the first hidden state (tag) in a sentence. Since the first word in a sentence has no predecessor, I follow the idea of assuming the most frequent tag that the word has been observed with in the training data as the most probable tag for this word.
2. Sate transition probabilities (a_{ij}), stored in a transition matrix, quantify the likelihood of observing one hidden state given the previous hidden state.
3. State emission probabilities (b_{ij}), stored in a confusion matrix, specify the probabilities of observing a particular state (word) while the HMM is in a certain hidden state.

When training a tagger in a supervised fashion, these parameters are estimated from the learning data. In fact, parameters estimation during training is a visible Markov process, because the surface pattern (words) and underlying MM (POS sequence) are fully observed. In contrast to that, the process of applying the trained MM to label unseen data truly represents a HMM, because the hidden sequence is unknown.

2.1 Algorithm for Implementing HMM

How can decoding be efficiently implemented? One classical strategy is the Viterbi algorithm [11]. The essential intuition behind the Viterbi algorithm and its main advantage are the reduction of the complexity of examining every full path through a trellis by recursively finding partial probabilities (δ) for the most likely path from one state to the next. A trellis represents the related search space; that is a matrix of

all hidden states and connections (transitions) between them along a sequence of observed states. The Viterbi algorithm requires three steps for searching and identifying one complete and most probable route through the trellis [6]:

Viterbi algorithm $\delta_j(t) = \max_X P(X, O, X_t = j \mid \mu)$

where $X = X_1 \ldots X_{t-1}$ and O = output sequence = $O_1 \ldots O_{t-1}$

1. Initialization $\delta_j(1) = \pi_j, 1 \le j \le N$
2. Induction $\delta_j(t+1) = \max_{1 \le i \le N} \delta_i(t)\, a_{ij}\, b_{ijot}, 1 \le j \le N$

   Store backtrace $\psi_j(t+1) = \arg\max_{1 \le i \le N} \delta_i\, a_{ij}\, b_{ijot}, 1 \le j \le N$

   where $\psi_j(t)$ = storage of node of incoming arc to most probable path
3. Termination and path (most likely tag sequence) readout (by backtracking)

$$\hat{X}_{T+1} = \arg\max_{1 \le i \le N} \delta_i(T+1)$$

$$\hat{X}_t = \psi_{\hat{X}_{T+1}}(t+1)$$

$$P(\hat{X}) = \max_{1 \le i \le N} \delta_i(T+1)$$

2.2 Looking at Viterbi's pieces through a magnifying glass

In this paper, I examine the performance of the Viterbi algorithm with respect to accuracy in more detail by identifying the partial accuracy gain that the different steps or components that the Viterbi algorithm involves account for. Specifically, I determine computationally how much of the total accuracy that the implementation of Viterbi approximation as specified above achieves can be attributed to the consideration of a) partial probabilities (δ), back pointers (ψ) and backtracking, b) initial, transition and confusion probabilities, and c) probabilities of single words.

Point a) represents the heart of the Viterbi algorithm. It is the combination of the forward algorithm, back pointers and backtracking that enables Viterbi to find the globally best path through a trellis. Further on, I refer to point a) as Viterbi.

Point b) represents the key idea of HMM, which is contained in the initialization and part of the induction step of the Viterbi algorithm. I suggest that the numeric difference between points a) and b) represents the accuracy gain that Viterbi can provide over the mere concatenation of tags that are chosen as the maximal products out of all combination of state transition and emission probabilities between subsequent words. In short, the difference between a) and b) represents the difference between a globally versus a locally maximal solution. Here I define local maximum as the outcome of induction that excludes δ from computation. Further on, I refer to point b) as HMM.

Point c) resembles the initialization and initial state probabilities as described earlier in this section; disregarding the impact of a word's predecessor on a word's POS and not making use of the relaxed independence assumption among words in MM. In HMM and Viterbi, the computation of point c) applies to every first word in a sentence as well as one word sentences. Further on, I refer to point c) as the Unigram Model (UM).

I suggest that knowledge of the contribution of points b) and c) to the total accuracy of Viterbi is relevant for people who need an intimate understanding of the

algorithm and its components, e.g. in order to fine-tune or modularly modify respective systems.

3 Experiment

3.1 Data

The data set used for training and validation is the tagged version of the Brown Corpus from the Penn Treebank 3 (PTB) corpus [7]. The PTB collection contains 2,499 news stories from over three years of the Wall Street Journal. Every word in the corpus is annotated with at least one out of 36 possible tags. I implemented a routine that collects the data, which is stored in 500 data files that are organized in 15 folders on a protected web site, and stores them locally on my system.

In cases where the authors of the PTB were uncertain about the best POS for a word, e.g. when a word was syntactically ambiguous, they assigned multiple tags in a non-standardized order [7]. The example *England-born/NNP|VBN* means that England-born might be a singular proper noun as well as past-participle verb. I performed several qualitative checks (human reasoning about the best out of the offered tags) on randomly drawn instances of this issue from PTB, which convinced me on the random order of multiple tags per word. Therefore, I decided to consistently consider only the first POS in all cases of tag indeterminacy.

3.2 Order of HMM

In a first order HMM, only one precedent tag is considered (n=1). This resembles the bi-gram model. One might expect N-grams of sizes larger than two to lead to more accurate predictions, because word sequences might depend not only on one, but multiple predecessors (e.g. *Department of Labor*). Previous research has shown that such an approach results in less and sparser training data due to the lack of local histories for the beginning of sentences [6]. This can pose a serious disadvantage if sentences in the training data are rather short on average, or if comas instead of sentence marks are considered as delimiters. The average sentence in the PTB 3 has a length of 19.8 words [9]. Thus, a shift from a first order HMM to a second order HMM would reduce the amount of training data by about 5.05%, and in addition to that make it sparser. For these reasons, I decided to work with a first-order HMM.

3.3 Smoothing

The implementation of the proposed system required cautious handling of small numbers and zero probabilities at various points. First, propagating and multiplying partial probabilities in the induction step of Viterbi led to number underflows. This issue can be handled during training: In order to eliminate this problem, I used the natural logs of the transition and emission probabilities, and changed the related multiplications into summations. Note that this problem does not apply to the other two algorithms that were implemented, because both of them disregard partial probabilities.

Second, words and state sequences that have not been observed in the training data, but do occur in the evaluation data, will cause

a) Zero probability in the induction step of Viterbi. As a result, an entire vertical column in the trellis (all δ for step i) would have zero probabilities, and thus the propagation of most probable paths would break.

b) Accuracy losses for all three models during evaluation. This is because "unknown" never matches the tag that the labeled comparison set suggests for a word. In an initial ten-fold validation test (described in detail in section 3.4), unknown words accounted for 3.94% of the words, with each of them contributing to an increase in the error rate.

Problems 2a) and 2b) depend on the test data: even a model trained on a humongous training set is likely to encounter unknown words when being applied to unseen data. This issue represents the downside of the time-invariance assumption. It occurs e.g. when new names for people, organizations or products are introduced or combined. To circumvent problems 2a) and b) I implemented two sets of strategies:

Zero probabilities for transition and emission are avoided by adding the words that are newly encountered during evaluation to the confusion matrix and tagging them as "UNKNOWN". This leads to emission probabilities different from zero for new words. Next, transition probabilities that involve the UNKNOWN tag have not been observed previously and therefore equal zero, thus stopping the propagation of the most likely paths through the trellis. To resolve this problem I assigned a small probability (minProb) to the affected transitions: P (t | t=UNKNOWN) = minProb and P(t=UNKNOWN | t) = minProb. For this application I chose a ln(minProb) = -100,000,000. This solution follows the Adding One strategy [2], which in addition to linear interpolation is a frequently applied smoothing technique in tagging [6].

Unknown words are passed to a post-processing routine that applies a set of rules in order to re-label unknown words with an actual POS. The best-performing unknown-word resolution techniques in tagging use information about the word's spelling [12][4]. Following this idea, I analyzed the unknown words that resulted from multiple evaluation runs in detail in order to find regularities in their association with certain POS. Based on this analysis, I implemented four types of orthographic rules for tagging unknown words. I assume these rules not only to be corpus-specific, but also to be of general applicability: Words involving a digit are tagged as numbers (CD). Capitalized words are tagged as singular proper nouns (NNS). Words ending in one out of 16 inflectional or derivational endings are tagged with the respective POS (e.g. *–ing* tagged as gerund (VBG)). Every remaining unknown word is labeled as singular or mass noun (NN).

3.4 Evaluation

In order to determine the accuracy of Viterbi (globally maximal solution), HMM (locally maximal solution), and UM (probabilistically maximal solution), multiple ten-fold cross validations were performed. The validation was implemented by first randomly breaking the full corpus (500 files, about 1.07 million words) into ten partitions. Then, nine folds (450 files, 965,828 words in one full ten-fold run) were used for training. Finally, I removed all tags from the remaining fold (50 files, 107,314 words in one run), used the three algorithms to tag the data in the tenth fold, compared the automatically assigned tags to the original labeling of the tenth fold, and recorded all deviations as errors. This procedure was repeated ten times[1] and the error rates were averaged.

Typically, taggers are evaluated by running the Gold Standard test and/ or comparing the results to a Unigram Baseline test [4]. The Gold Standard represents the performance measure - the accuracy rates for Viterbi and HMM - by determining the portion of tagged words that the tagger and a human-labeled validation set agree

[1] Elapsed running time for training all three models, performing a full ten-fold run, and reporting averaged error rates is about 10.5 minutes on an IBM T60p Intel CoreDuo2.16 GHz with 2 GB of RAM.

upon. The Unigram Baseline test follows the simplifying assumption that every word and its POS are context-independent, thus associating every word with the tag that the word has been most frequently observed with in the training set. In this project, the UM represents this baseline measure.

The experimental results show that on average, Viterbi achieves 93.9% accuracy, HMM 93.5%, and UM 88.9%. These numbers do not match, but closely approximate the best accuracy rates reported for HMM- and Viterbi taggers (96% to 97%), as well as for Baseline taggers (90-91%) [4].

Figure 1: Accuracy per Algorithm (runs sorted by increasing error)

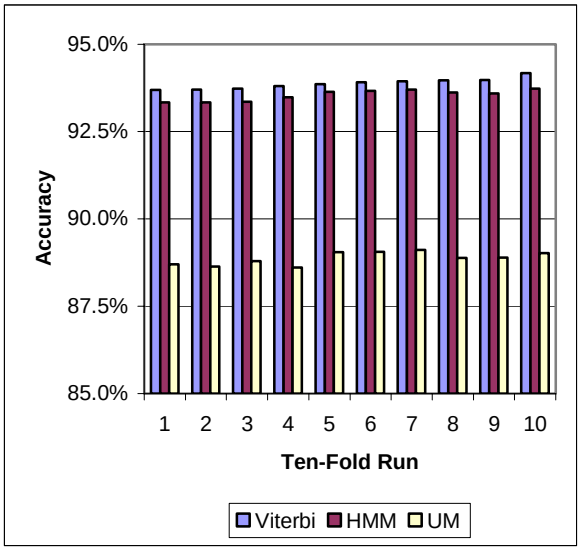
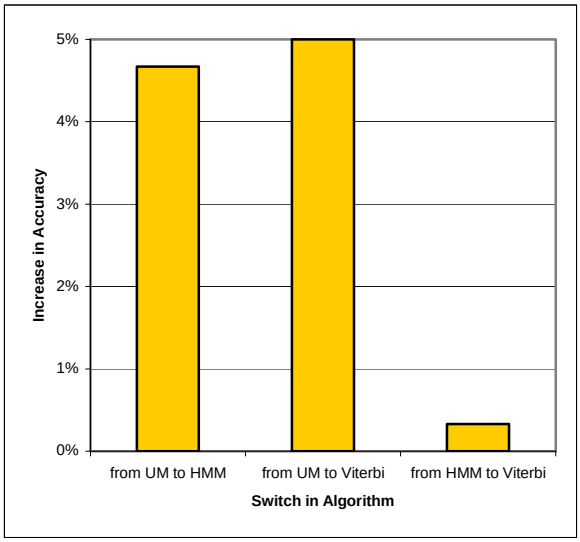


Table 1: Statistics on Accuracy per Algorithms

	Viterbi	HMM	UM
Average	93.87%	93.54%	88.88%
Min	93.69%	93.34%	88.61%
Max	94.18%	93.72%	89.11%
St. Dev.	0.15%	0.15%	0.18%

Figure 2: Partial Accuracy Gain by Algorithm



How high is the partial accuracy gain that can be attributed to the different components of the Viterbi algorithm? The findings indicate that the UM alone achieves almost 89% accuracy (Figure 2). Building on top of that, the switches from UM to HMM (+4.67%) and from UM to Viterbi (+5.0%) seem significantly beneficial. In contrast to that, the step from HMM to Viterbi further increased accuracy by only 0.33%.

In order to further investigate the partial impact of computational routines on accuracy, I disabled the post-processor, which takes care of tagging unknown words, and ran a ten-fold validation. The outcome, as shown in Table 2, reveals that unknown words account for less than half a percent of the error rate per algorithm.

Table 2: Accuracy Loss caused by Not Handling of Unknown Words

Algorithm	Accuracy without Post Processing	Loss due to Unknown Words
Viterbi	93.49%	0.38%
HMM	93.25%	0.29%
UM	88.48%	0.40%

Table 3: Averaged Accuracy for 20-fold validation

	Viterbi	HMM	UM
Average	93.95%	93.66%	88.92%
Min	93.46%	93.24%	88.28%
Max	94.56%	94.35%	89.54%
St. Dev.	0.34%	0.35%	0.36%
Gain over Ten-Fold			
Average	0.08%	0.12%	0.05%
Min	0.23%	0.10%	0.33%
Max	0.38%	0.63%	0.43%
St. Dev.	0.19%	0.19%	0.18%

Finally, I aimed to determine how much the size of the training set impacts accuracy. One might assume that in tagging, a larger training set leads to a lower error rate, because more words and combinations of hidden and observed states can be learned. To empirically test for this hypothesis, I performed ten 20-fold validations (corpus split into 97.5% training data and 2.5% test data). The averaged results (Table 3) suggest a small accuracy gain (0.05% to 0.12%, depending on algorithm), while the variation of the results (standard deviation) increases.

4 Conclusions

The algorithms that were implemented, tested and compared performed reasonably well on tagging unseen texts that were randomly and independently drawn from the same corpus as the training set. What does *reasonably well* mean? Assuming that an average sentence in the PTB corpus is 19.8 words long [8], the tagger, on average, would mislabel 1.21 (Viterbi), 1.28 (HMM) or 2.21 (UM) words per sentence.

The experimental quantification of the partial information gain due to each algorithm revealed that UM, a simple probabilistic model used for initializing state probabilities in HMM and Viterbi, achieves a respectably high accuracy of 88.9%. However, UM is clearly outperformed by HMM, because this switch between algorithms leads to an accuracy gain of 4.7%. Viterbi results in the highest accuracy rate (93.9%), but provides only a slight accuracy gain of 0.33% over HMM.

The empiric identification of the impact of handling unknown words on accuracy rates leaves us with the insight that implementing reasonable strategies for re-labeling unknown words as POS can be more beneficial than the upgrade from HMM to Viterbi. The average gains in tagging accuracy due to an increase in the size of the training set were lower for all three algorithms than the contributions by the handling of unknown words. Furthermore, for larger training sets, an increase in the variation of the results has been observed. This finding might suggest a careful

consideration of this trade-off to researchers before they commit to this strategy. Especially for UM, the enlarged training set only led to a marginal accuracy gain of 0.05%, which might indicate that the performance limit of this algorithm has been asymptotically approached by this point.

Several limitations of this project should be tackled in future work. First, the models were trained and evaluated on one specific data set. Even though this corpus contains more than a million observations, it still reflects a certain time period, style (journalistic writing) and range of domains (news paper articles). Applying the models to data that differs from these features is likely to result in lower accuracy rates as achieved herein. Second, the focus of the project (accuracy gain due to different algorithms, handling of unknown words, and training set size) did not encompass testing of MM of a higher order. For data sets with lengthy sentences, e.g. academic writing, trigrams or larger N-grams might further improve tagging accuracy. Finally, all tested algorithms are stochastic taggers; therefore a comparison to accuracy rates achieved with rule- or transformation-based systems could be valuable.

Acknowledgements

I am grateful to Alex Rudnicky from CMU for providing the training data to me and to Yifen Huang, CMU, for discussing the project with me.

References

[1] Baum, L. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities* 3: 1-8.

[2] Church, K. (1988). A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *2nd Conference on Applied Natural Language Processing*, Austin , TX, 136-143.

[3] DeRose, S. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14:31-39.

[4] Jurafsky, D. & Martin, J.H. (2000). *Speech and Language Processing*. Upper Saddle River, N.J.: Prentice Hall.

[5] Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6: 225-242.

[6] Manning, C. & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press. Cambridge, MA.

[7] Mitchell P.M. Santorini, B., Marcinkiewicz, M.A. & Taylor, A. (1999). *Advances Treebank 3.* Linguistic Data Consortium, Philadelphia.

[8] Rabiner L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech. Recognition. *Proc. IEEE* 77(2): 257-285.

[9] Riezler, S. et al. (2002). Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. *Pro. 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA, USA.

[10] Stolz, W.S., Tannenbaum, P.H. & Carstensen, F.V. (1965). Stochastic Approach to the Grammatical Coding of English. *Communications of the ACM* 8: 399–405.

[11] Viterbi, A. J. (1967). Error bounds for convolutional codes and asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory* 13: 260-269.

[12] Weischedel, R., Meter, M., Schwartz, R., Ramshaw, L., & Palmucci, J. (1993). Coping with. ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19 (2): 359-382.